Evaluating the Productivity of a Multicore Architecture

Jeremy Kepner and Nadya Bliss

MIT Lincoln Laboratory

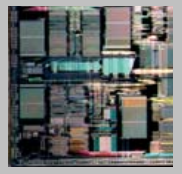
HPEC 2008

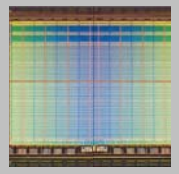
This work is sponsored by the Department of Defense under Air Force Contract FA8721-05-C-0002. Opinions, interpretations, conclusions, and recommendations are those of the author and are not necessarily endorsed by the United States Government.

Outline

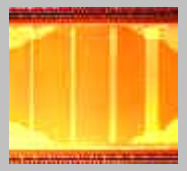
- **Parallel Design**
  - *Architecture Buffet*
  - *Programming Buffet*
  - *Productivity Assessment*
- Programming Models
- Architectures
- Productivity Results
- Summary

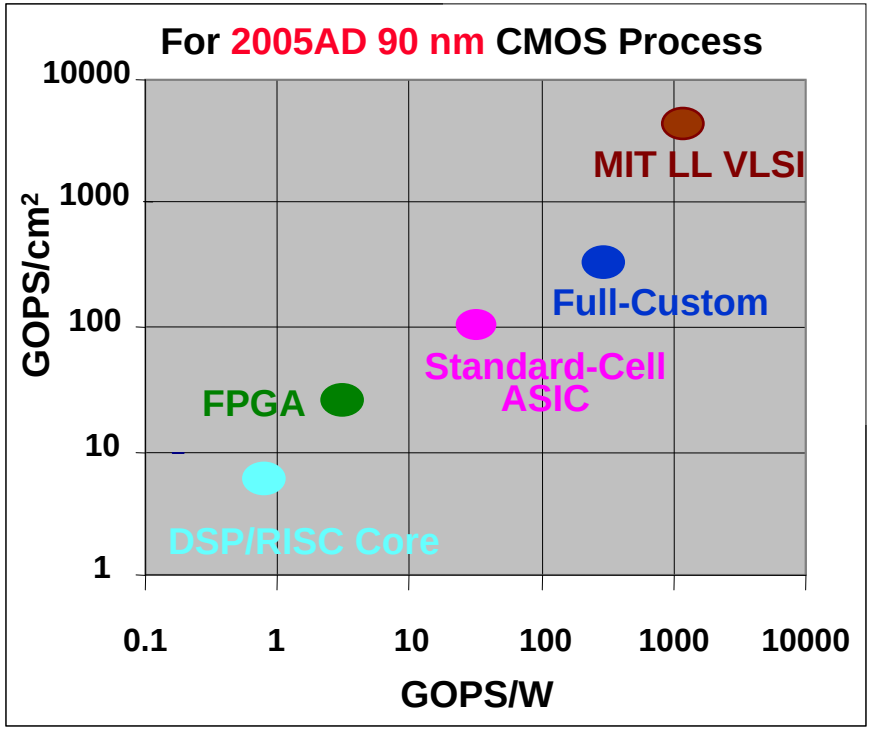
Signal Processor Devices

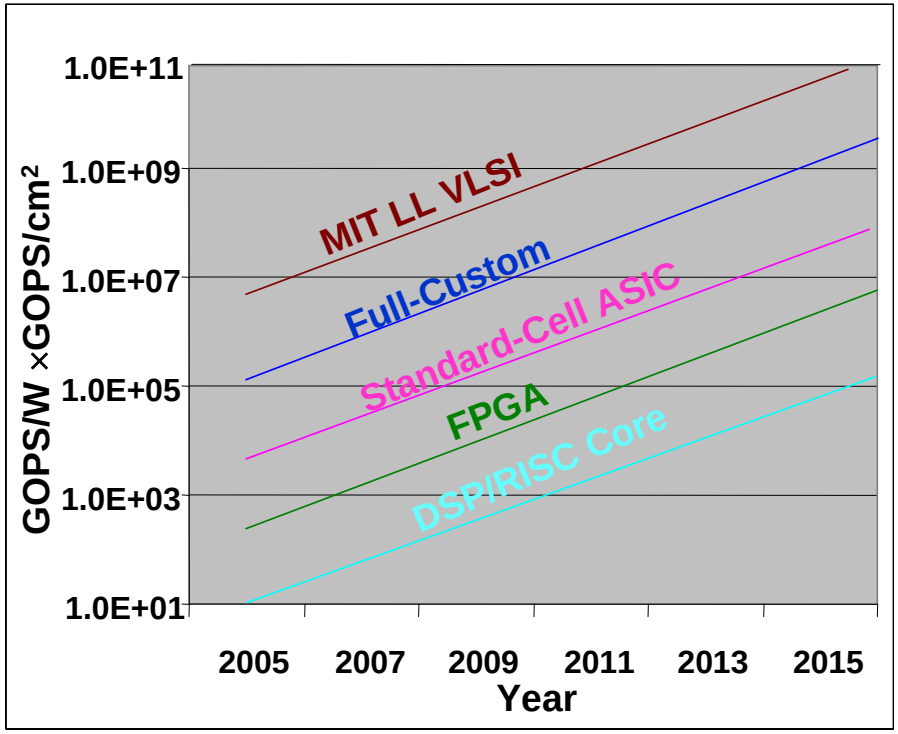
DSP/RISC Core 

FPGA 

ASIC 

Full Custom 





- **Wide range of device technologies for signal processing systems**
- **Each has their own tradeoffs. *How do we choose?***

Multicore Processor Buffet

	Homogeneous	Heterogeneous
Short Vector	• Intel Duo/Duo • AMD Opteron • IBM PowerX • Sun Niagara • IBM Blue Gene • Broadcom • Tilera	• IBM Cell • Intel Polaris
Long Vector	• Cray XT • Cray XMT • Clearspeed	• nVidia • ATI

- Wide range of programmable multicore processors
- Each has their own tradeoffs. *How do we choose?*

Multicore Programming Buffet

	Flat	Hierarchical
word	• pThreads • StreamIt • UPC • CAF	• Cilk • CUDA • ALPH • MCF • Sequouia
object	• VSIPL++ • GA++ • pMatlab • StarP	• PVTOL • pMatlabXVM

- Wide range of multicore programming environments
- Each has their own tradeoffs. *How do we choose?*

Performance vs Effort

Style	Example	Granularity	Training	Effort	Performance per Watt
Graphical	Spreadsheet	Module	Low	1/30	1/100
Domain Language	Matlab, Maple, IDL	Array	Low	1/10	1/5
Object Oriented	Java, C++	Object	Medium	1/3	1/1.1
Procedural Library	VSIPL, BLAS	Structure	Medium	2/3	1/1.05
Procedural Language	C, Fortran	Word	Medium	1	1
Assembly	x86, PowerPC	Register	High	3	2
Gate Array	VHDL	Gate	High	10	10
Standard Cell		Cell	High	30	100
Custom VLSI		Transistor	High	100	1000

Programmable Multicore
(this talk)

- Applications can be implemented with a variety of interfaces
- Clear tradeoff between effort (3000x) and performance (100,000x)
  - Translates into mission capability vs mission schedule

Assessment Approach

- "Write" benchmarks in many programming environments on different multicore architectures
- Compare performance/watt and relative effort to serial C

Outline

- Parallel Design
- **Programming Models**
  - *Environment features*
  - *Estimates*
  - *Performance Complexity*
- Architectures
- Productivity Results
- Summary

Programming Environment Features

Technology	UPC	F2008	GA++	PVL	VSIPL	PVTOL	Titanium	StarP	pMatlab	DCT	Chapel	X10	Fortress
Organization	Std Body	Std Body	DOE PNNL	Lincoln	Std Body	Lincoln	UC Berkeley	ISC	Lincoln	Math-works	Cray	IBM	Sun
Sponsor	DoD	DOE SC	DOE	Navy	DoD HPCMP		DOE, NSF	DoD	DARPA		DARPA	DARPA	DARPA
Type	Lang Ext	Lang Ext	Library	Library	Library	Library	New Lang	Library	Library	Library	New Lang	New Lang	New Lang
Base Lang	C	Fortran	C++	C++	C++	C++	Java	Matlab	Matlab	Matlab	ZPL	Java	HPF
Precursors		CAF		STAPL, POOMA	PVL, POOMA	VSIPL++, pMatlab		pMatlab	PVL, StarP	pMatlab, StarP			
Real Apps	2001	2001	1998	2000	2004	~2007		2002	2003	2005			
Data Parallel	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Block-cyclic	1D		ND blk	2D	2D	Y	ND	2D	4D	1D	ND	ND	
Atomic			Y									Y	Y
Threads	Y		Y								Y	Y	Y
Task Parallel			Y	Y	Y	Y	Y		Y		Y	Y	
Pipelines			Y	Y		Y			Y				
Hier. arrays						Y	Y		Y		Y	Y	Y
Automap				Y		Y			Y				
Sparse							?	Y	Y	Y	Y	?	?
FPGA IO					Y	Y							

- **Too many environments with too many features to assess individually**
- **Decompose into general classes**
  - **Serial programming environment**
  - **Parallel programming model**
- **Assess only relevant serial environment and parallel model pairs**

Dimensions of Programmability

- **Performance**
  - The performance of the code on the architecture
  - Measured in: flops/sec, Bytes/sec, GUPS, …
- **Effort**
  - Coding effort required to obtain a certain level of performance
  - Measured in: programmer-days, lines-of-code, function points,
- **Expertise**
  - Skill level of programmer required to obtain a certain level of performance
  - Measured in: degree, years of experience, multi-disciplinary knowledge required, …
- **Portability**
  - Coding effort required to port code from one architecture to the next and achieve a certain level of performance
  - Measured in: programmer-days, lines-of-code, function points, …)
- **Baseline**
  - All quantities are relative to some baseline environment
  - Serial C on a single core x86 workstation, cluster, multi-core, …

Serial Programming Environments

Programming Language	Assembly	SIMD (C+AltiVec)	Procedural (ANSI C)	Objects (C++, Java)	High Level Languages (Matlab)
Performance Efficiency	0.8	0.5	0.2	0.15	0.05
Relative Code Size	10	3	1	1/3	1/10
Effort/Line-of-Code	4 hour	2 hour	1 hour	20 min	10 min
Portability	Zero	Low	Very High	High	Low
Granularity	Word	Multi-word	Multi-word	Object	Array

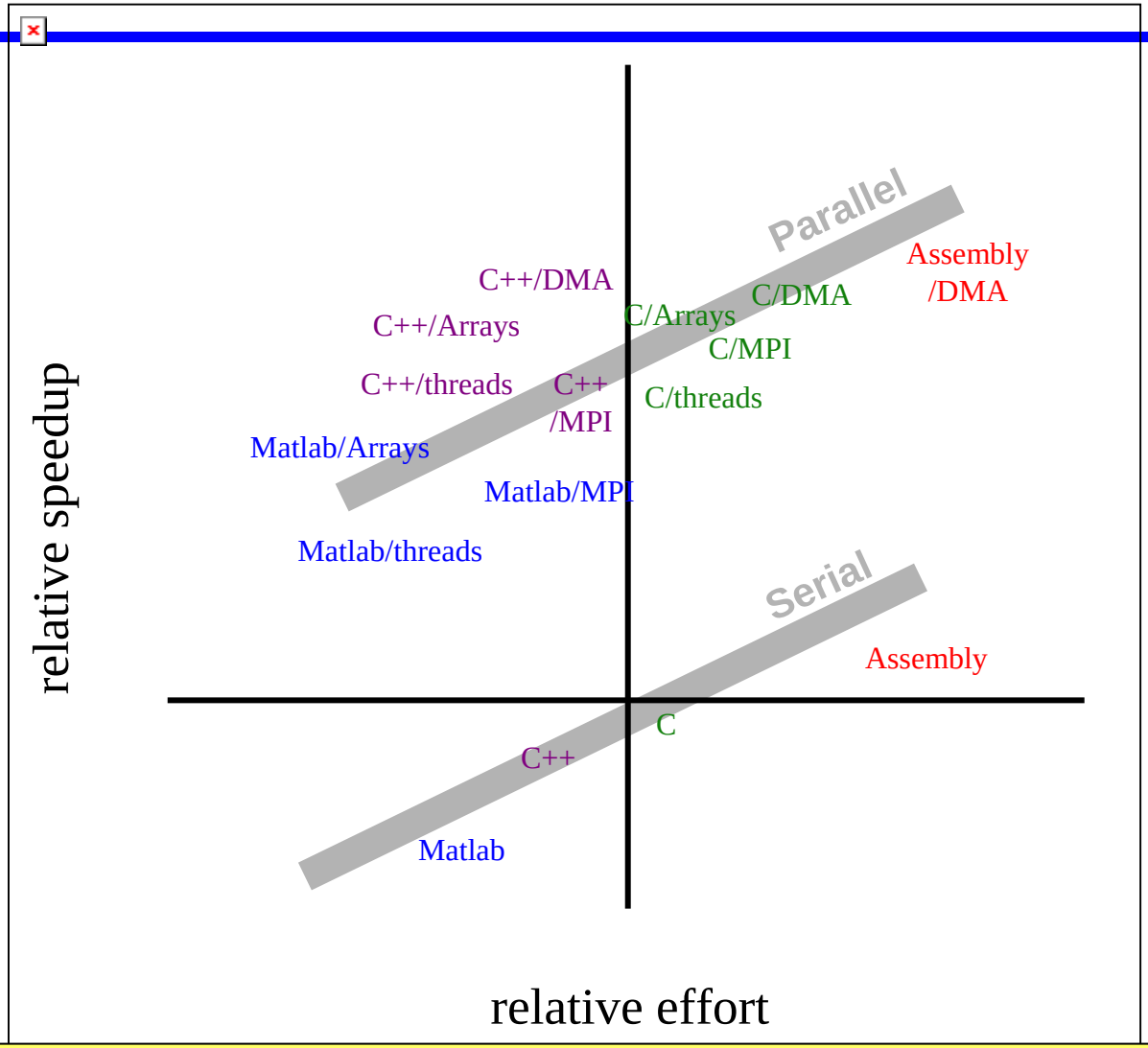
- OO High Level Languages are the current desktop state-of-the practice :-)
- Assembly/SIMD are the current multi-core state-of-the-practice :-(
- Single core programming environments span 10x performance and 100x relative code size

Parallel Programming Environments

Approach	Direct Memory Access (DMA)	Message Passing (MPI)	Threads (OpenMP)	Recursive Threads (Cilk)	PGAS (UPC, VSIPL++)	Hierarchical PGAS (PVTOL, HPCS)
Performance Efficiency	0.8	0.5	0.2	0.4	0.5	0.5
Relative Code Size	10	3	1	1/3	1/10	1/10
Effort/Line-of-Code	Very High	High	Medium	High	Medium	High
Portability	Zero	Very High	High	Medium	Medium	TBD
Granularity	Word	Multi-word	Word	Array	Array	Array

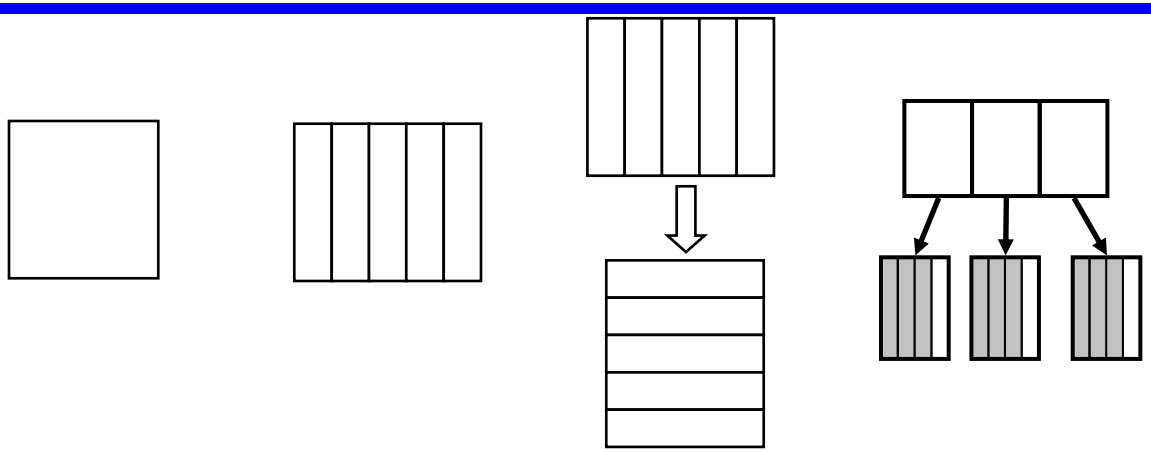
- Message passing and threads are the current desktop state-of-the practice :-|
- DMA is the current multi-core state-of-the-practice :-(
- Parallel programming environments span 4x performance and 100x relative code size

Canonical 100 CPU Cluster Estimates



- **Programming environments form regions around serial environment**

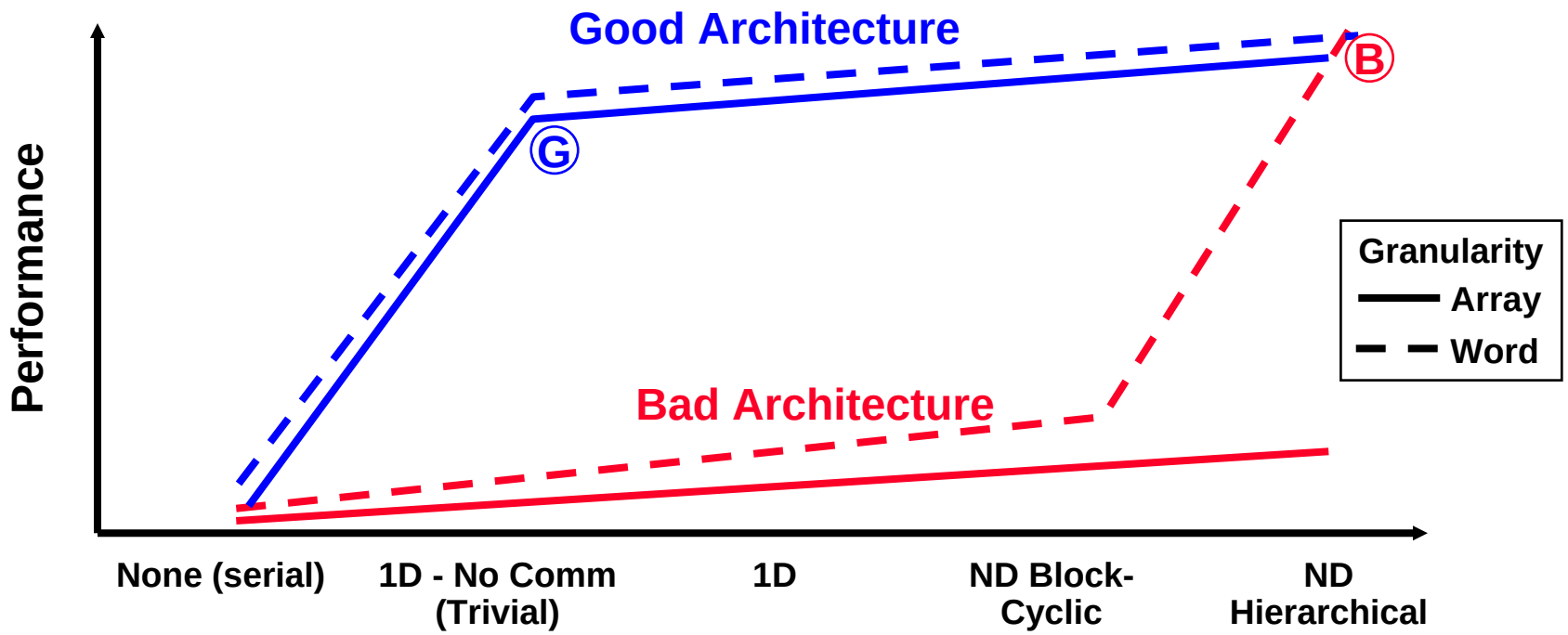
Relevant Serial Environments and Parallel Models



Partitioning Scheme	Serial	Multi-Threaded	Distributed Arrays	Hierarchical Arrays	Assembly + DMA
fraction of programmers	1	0.95	0.50	0.10	0.05
Relative Code Size	1	1.1	1.5	2	10
"Difficulty"	1	1.15	3	20	200

- Focus on a subset of relevant programming environments
  - C/C++ + serial, threads, distributed arrays, hierarchical arrays
  - Assembly + DMA
- "Difficulty" = (relative code size) / (fraction of programmers)

Performance Complexity

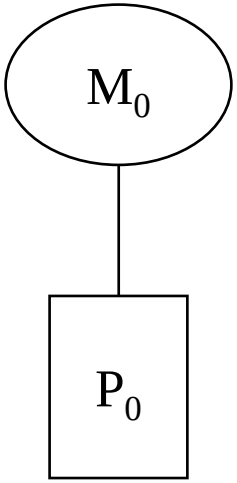


- Performance complexity (Strohmeier/LBNL) compares performance as a function of the programming model
- In above graph, point "G" is ~100x easier to program than point "B"

Outline

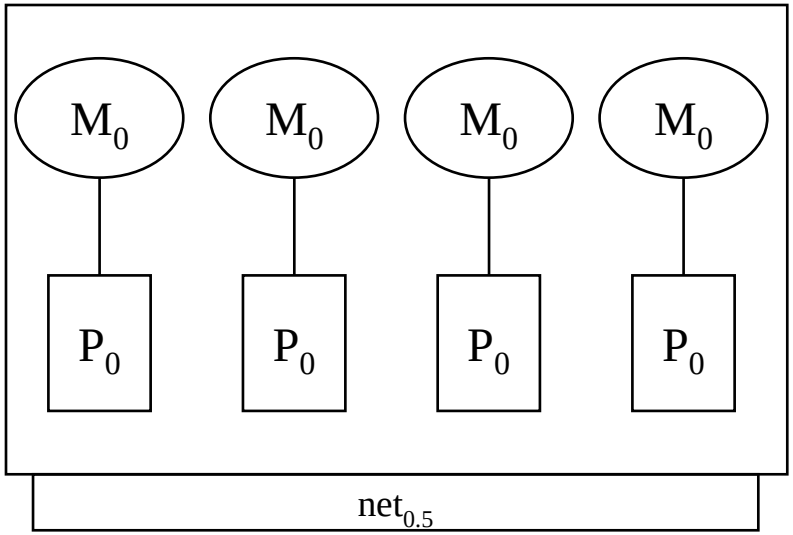
- Parallel Design
- Programming Models
- **Architectures**
  - *Kuck Diagram*
  - *Homogeneous UMA*
  - *Heterogeneous NUMA*
  - *Benchmarks*
- Productivity Results
- Summary

Single Processor Kuck Diagram



- Processors denoted by boxes
- Memory denoted by ovals
- Lines connected associated processors and memories
- Subscript denotes level in the memory hierarchy

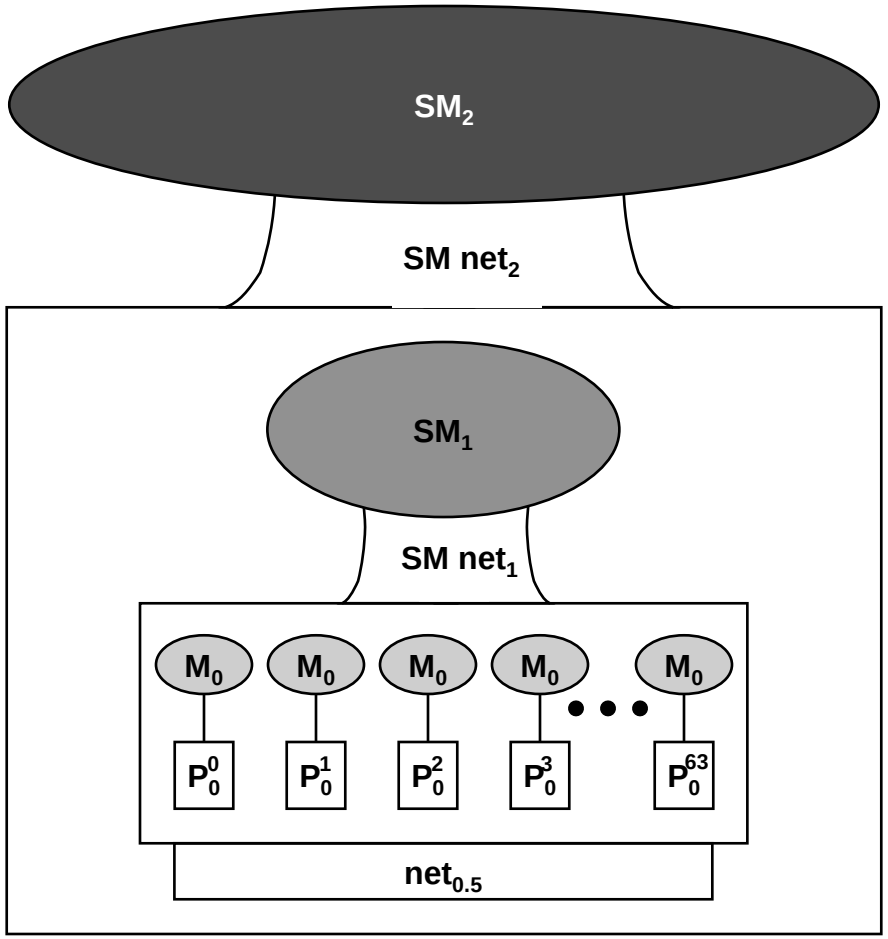
Parallel Kuck Diagram



- **Replicates serial processors**
- net **denotes network connecting memories at a level in the hierarchy (incremented by 0.5)**

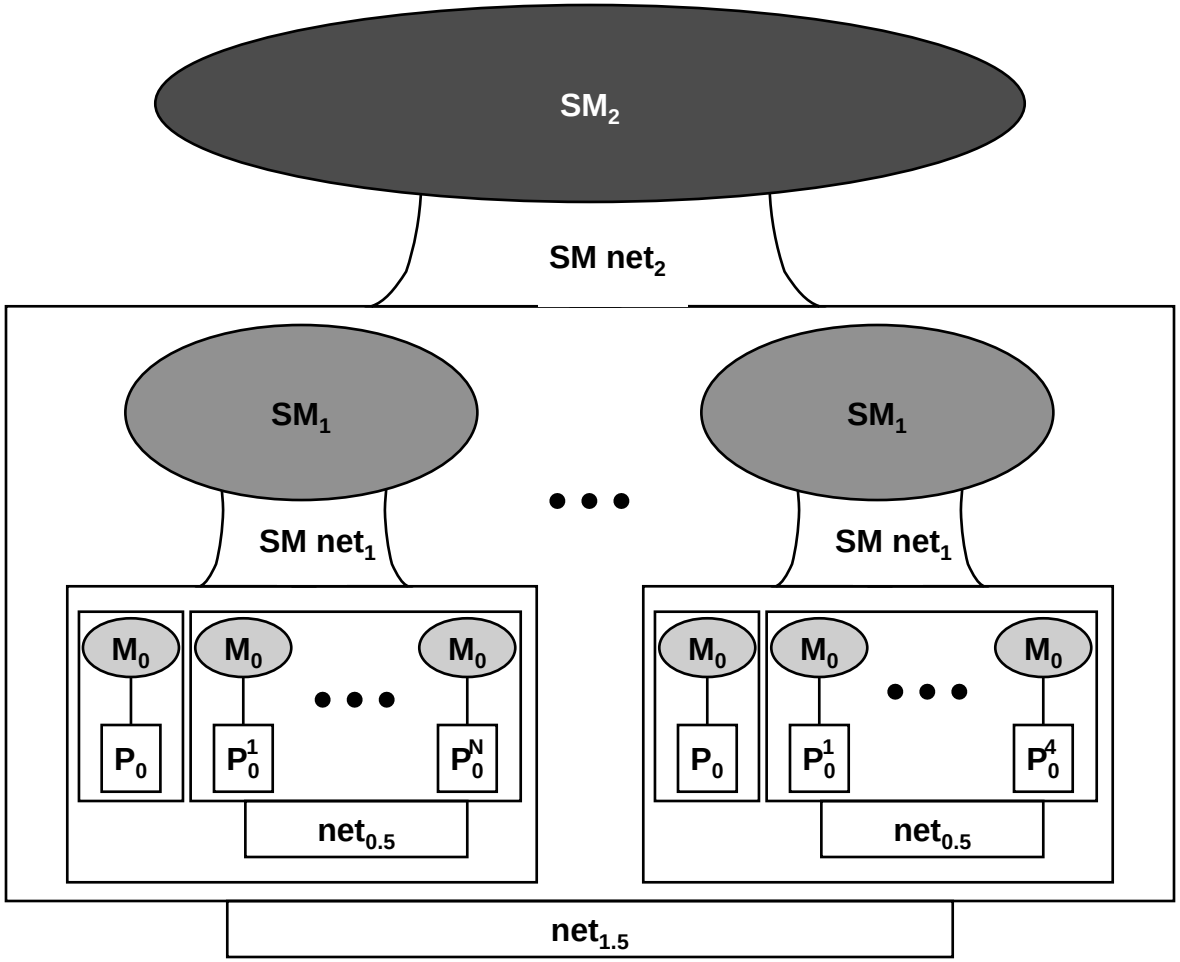
Multicore Architecture 1: Homogeneous

- Off-chip: 1 (all cores have UMA access to off-chip memory)
- On-chip: 1 (all cores have UMA access to on-chip 3D memory)
- Core: N_{core} (each core has its own cache)

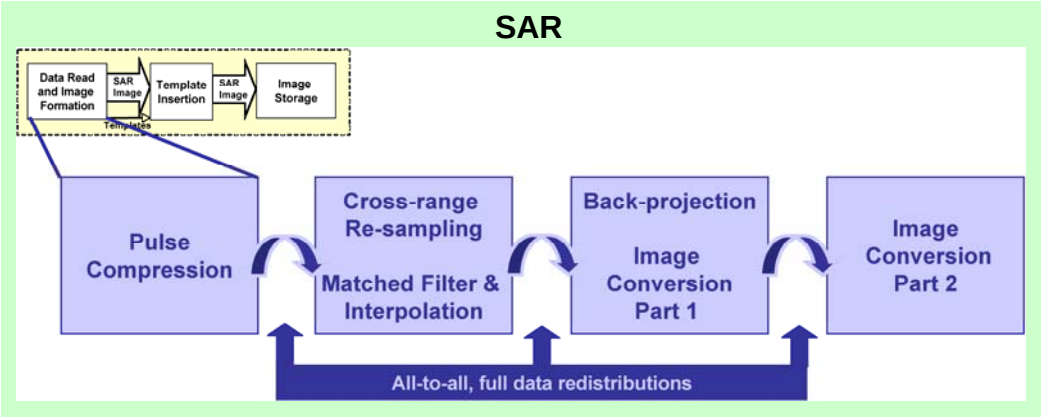


Multicore Architecture 2: Heterogeneous

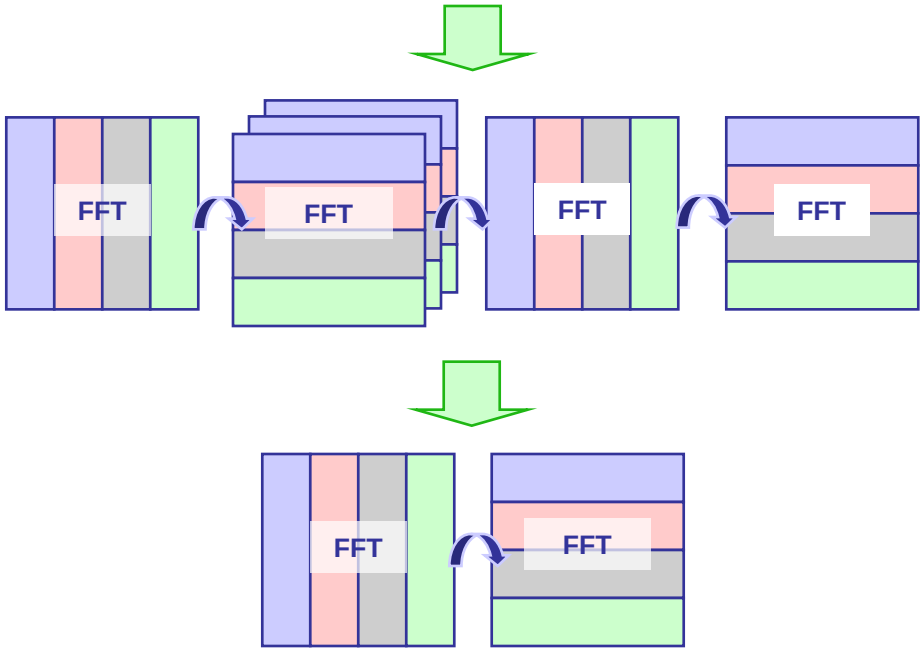
- Off-chip: 1 (all supercores have UMA access to off-chip memory)
- On-chip: N (sub-cores share a bank of on-chip 3D memory and 1 control processor)
- Core: N_{core} (each core has its own local store)



HPC Challenge SAR benchmark (2D FFT)



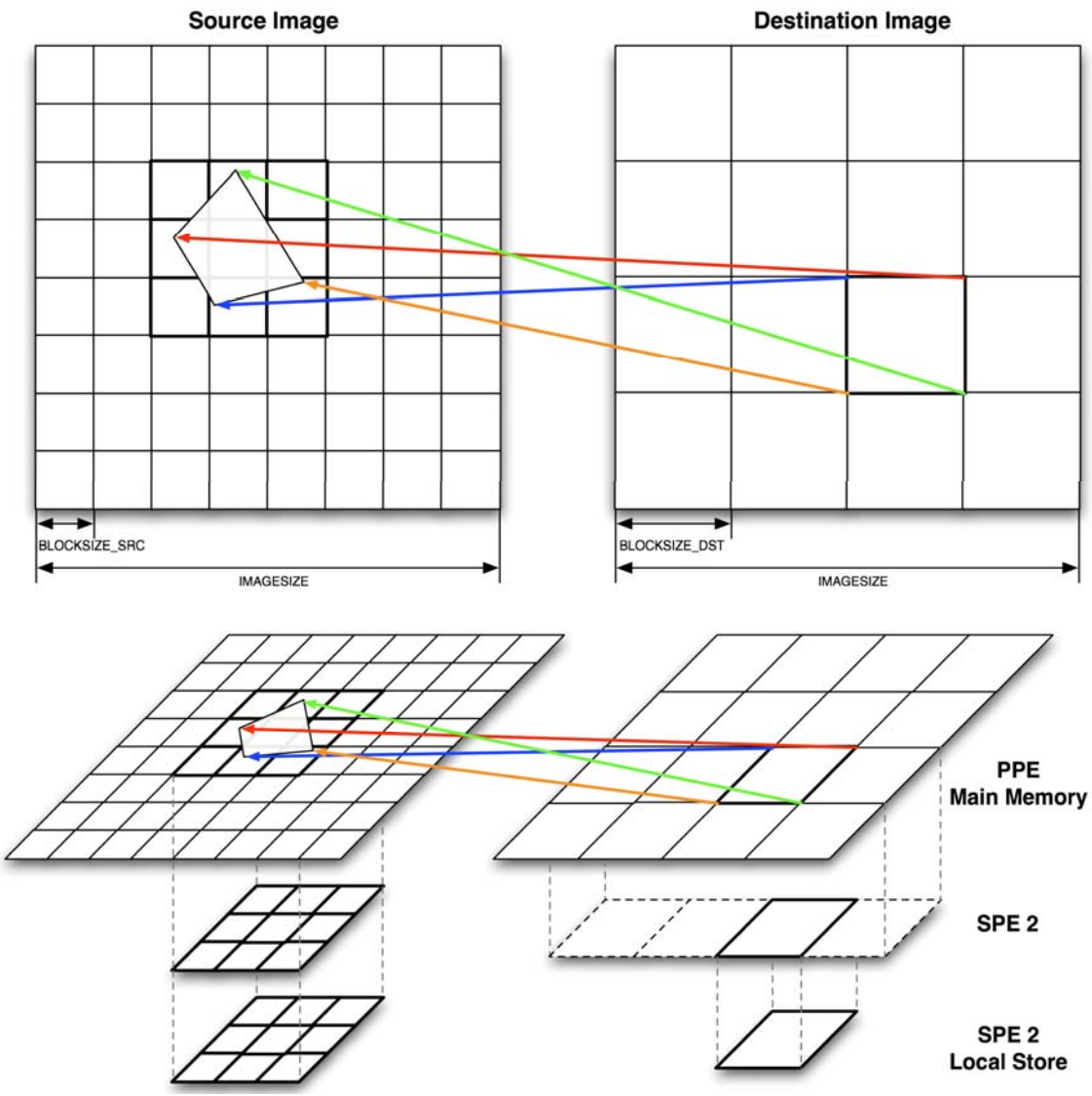
- 2D FFT (with a full all-to-all corner turn) is a common operation in SAR and other signal processing
- Operation is complex enough to highlight programmability issues




```
%MATLAB Code

A = complex(rand(N,M), ...
        ...rand(N,M));
%FFT along columns
B = fft(A, [], 1);
%FFT along rows
C = fft(B, [], 2);
```

Projective Transform



- Canonical kernel in image processing applications
- Takes advantage of cache on single core processors
- Takes advantage of multiple cores
- Results in regular distributions of both source and destination images

Outline

- Parallel Design
- Programming Models
- Architectures
- **Productivity Results**
  - *Implementations*
  - *Performance vs Effort*
  - *Productivity vs Model*
- Summary

Case 1: Serial Implementation

CODE

```
A = complex(rand(N,M), rand(N,M));

//FFT along columns
for j=1:M
 A(:,j) = fft(A(:,j));
end

//FFT along rows
for i=1:N
 A(i,:) = fft(A(i,:));
end
```

Heterogeneous Performance

Execution
- This program will run on a single control processor

Memory
- Only off chip memory will be used

NOTES

- Single threaded program
- Complexity: LOW
- Initial implementation to get the code running on a system
- No parallel programming expertise required
- Users capable of writing this program: 100%

Homogeneous Performance

Execution
- This program will run on a single core

Memory
- Off chip, on chip chache, and local cache will be used

Case 2: Multi-Threaded Implementation

CODE

```
A = complex(rand(N,M), rand(N,M));
#pragma omp parallel ...
//FFT along columns
for j=1:M
 A(:,j) = fft(A(:,j));
end
#pragma omp parallel ...
//FFT along rows
for i=1:N
 A(i,:) = fft(A(i,:));
end
```

Heterogeneous Performance

Execution
- This program will run on a all control processors

Memory
- Only off chip memory will be used
- Poor locality will cause cause a memory bottleneck

NOTES

- Multi-threaded program: each thread operated on a single column (row) of the matrix
- Complexity: LOW
- Minimal parallel programming expertise required
- Users capable of writing this program: 99%

Homogeneous Performance

Execution
- This program will run on all cores

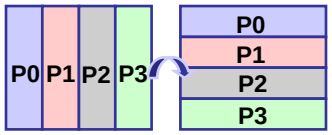
Memory
- Off chip memory, on chip cache, and local cache will be used
- Poor locality will cause a memory bottleneck

Case 3: Parallel 1D Block Implementation

CODE

```
mapA = map([1 36], {}, [0:35]); //column map
mapB = map([36 1], {}, [0:35]); //row map
A = complex(rand(N,M,mapA), rand(N,M,mapA));
B = complex(zeros(N,M,mapB), rand(N,M,mapB));
//Get local indices
myJ = get_local_ind(A);
myI = get_local_ind(B);
//FFT along columns
for j=1:length(myJ)
  A.local(:,j) = fft(A.local(:,j));
end
B(:,:) = A; //corner turn
//FFT along rows
for i=1:length(myI)
  B.local(i,:) = fft(B.local(i,:));
end
```

Distribution onto 4 processors



Heterogeneous Performance

Execution
- This program will run on all control processors

Memory
- Only off chip memory will be used

NOTES

- Explicitly parallel program using 1D block distribution
- Complexity: MEDIUM
- Parallel programming expertise required, particularly for understanding data distribution
- Users capable of writing this program: 75%

Homogeneous Performance

Execution
- This program will run on all cores

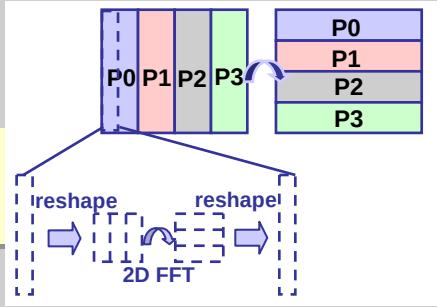
Memory
- Off chip memory, on chip cache, and local cache will be used
- Better locality will decrease memory bottleneck

Case 4: Parallel 1D Block Hierarchical Implementation

CODE

```
mapHcol = map([1 8], {}, [0:7]); //col hierarchical map
mapHrow = map([8 1], {}, [0:7]); //row hierarchical map
mapH = map([0:7]); //base hierarchical map
mapA = map([1 36], {}, [0:35], mapH); //column map
mapB = map([36 1], {}, [0:35], mapH); //row map
A = complex(rand(N,M,mapA), rand(N,M,mapA));
B = complex(zeros(N,M,mapB), rand(N,M,mapB));
//Get local indices
myJ = get_local_ind(A);
myI = get_local_ind(B);
//FFT along columns
for j=1:length(myJ)
  temp = A.local(:,j); //get local col
  temp = reshape(temp); //reshape col into matrix
  alocal = zeros(size(temp_col), mapHcol);
  blocal = zeros(size(temp_col), mapHrow);
  alocal(:,:) = temp; //distrbute col to fit into SPE/cache
  myHj = get_local_ind(alocal);
  for jj = length(myHj)
     alocal.local(:,jj) = fft(alocal.local(:,jj));
  end
  blocal(:,:) = alocal; //corner turn that fits into SPE/cache
  myHi = get_local_ind(blocal);
  for ii = length(myHi)
     blocal.local(ii,:) = fft(blocal.local(ii,:);
  end
  temp = reshape(blocal); //reshape matrix into column
  A.local = temp; //store result
end
B(:,:) = A; //corner turn
//FFT along rows ...
```

- Complexity: HIGH
- Users capable of writing this program: <20%



Heterogeneous Performance

Execution
- This program will run on all cores

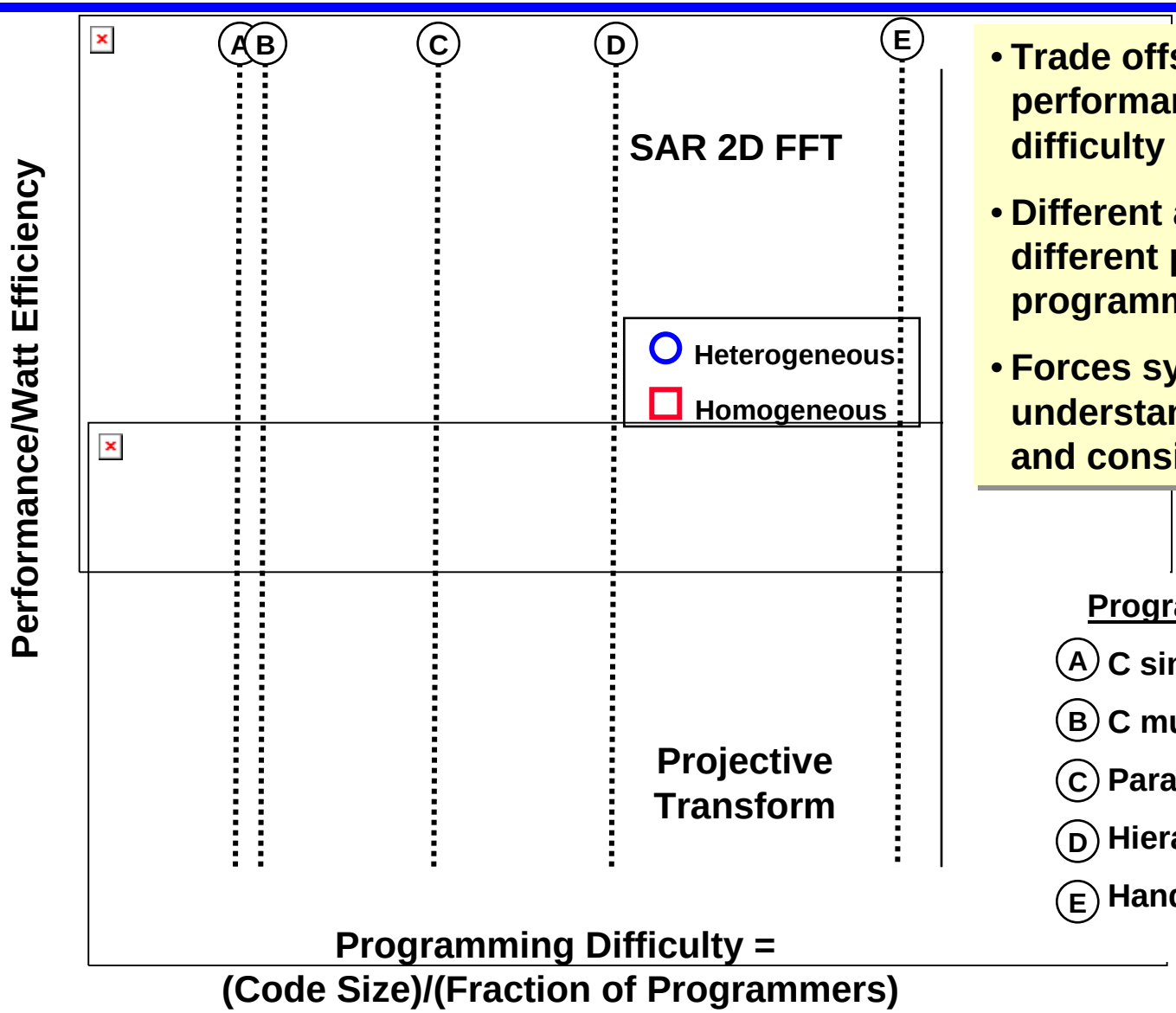
Memory
- Off chip, on-chip, and local store memory will be used
- Hierarchical arrays allow detailed management of memory bandwidth

Homogeneous Performance

Execution
- This program will run on all cores

Memory
- Off chip, on chip cache, and local cache will be used
- Caches prevent detailed management of memory bandwdith

Performance/Watt vs Effort



- Trade offs exist between performance and programming difficulty
- Different architectures enable different performance and programming capabilities
- Forces system architects to understand device implications and consider programmability

Programming Models

(A) C single threaded

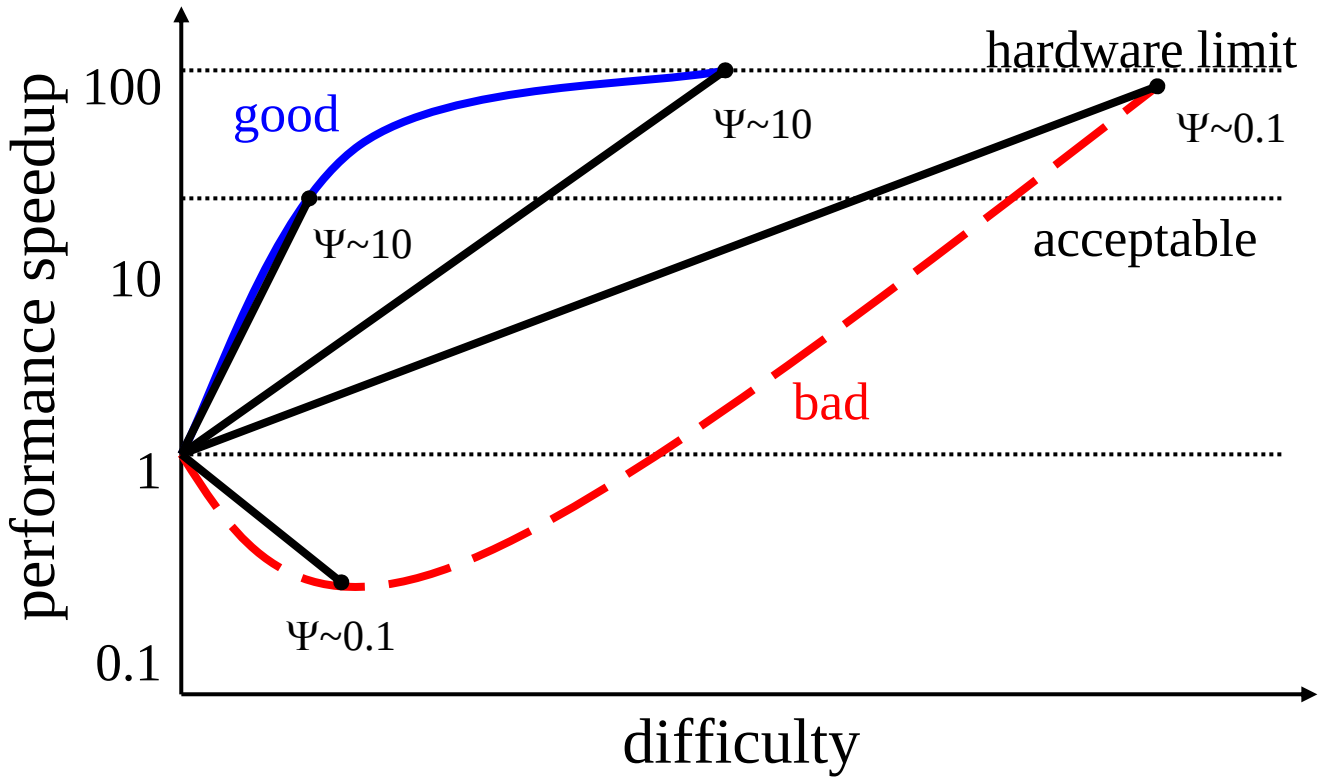
(B) C multi-threaded

(C) Parallel Arrays

(D) Hierarchical Arrays

(E) Hand Coded Assembly

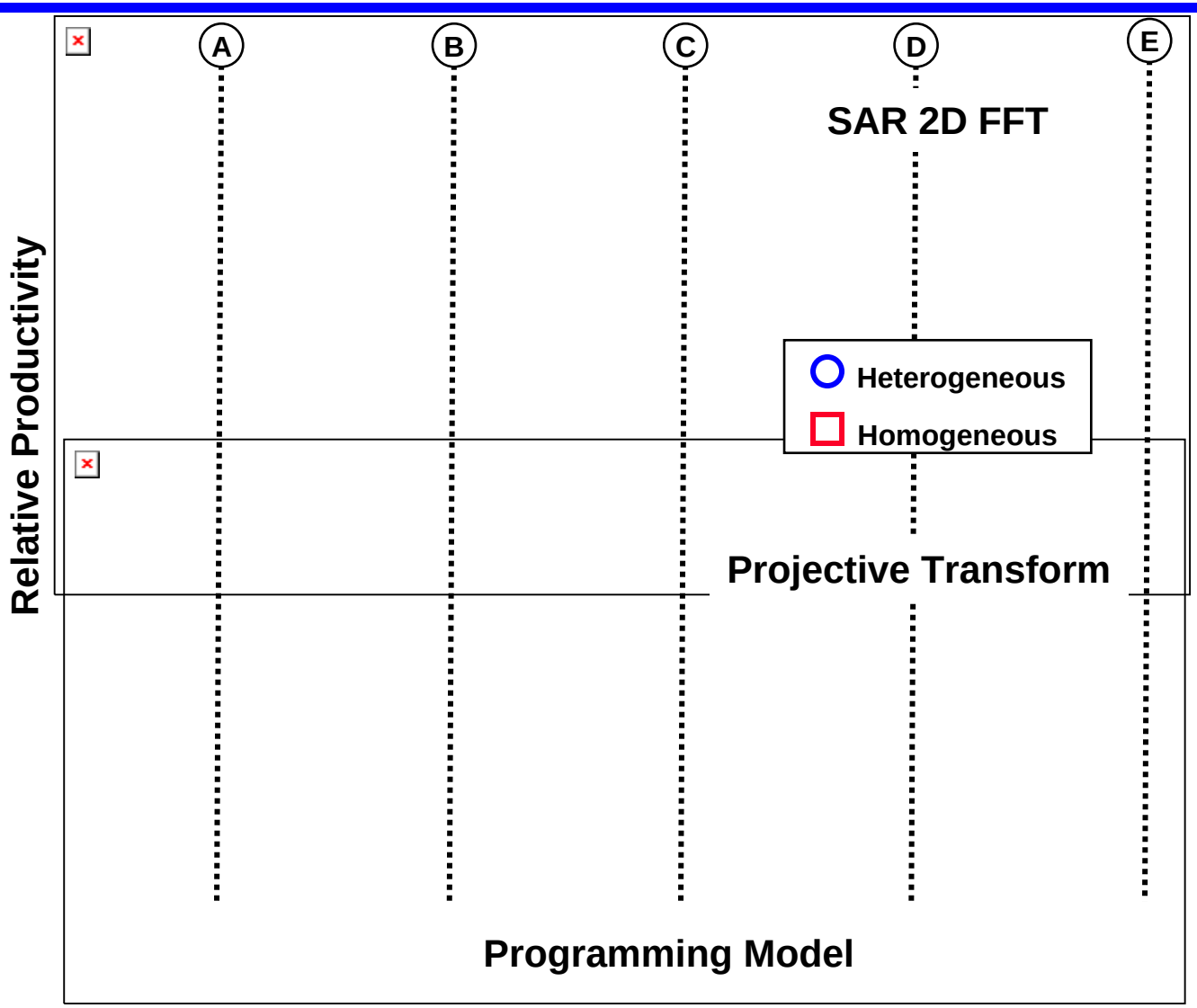
Defining Productivity



- **Productivity is a ratio between utility to cost**
- **From the programmer perspective this is proportional to performance over difficulty**

Productivity

Productivity vs Programming Model



- Productivity varies with architecture and application
- Homogeneous: threads or parallel arrays
- Heterogeneous: hierarchical arrays

Programming Models

- (A) C single threaded
- (B) C multi-threaded
- (C) Parallel Arrays
- (D) Hierarchical Arrays
- (E) Hand Coded Assembly

Summary

- Many multicore processors are available
- Many multicore programming environments are available
- Assessing which approaches are best for which architectures is difficult
- Our approach
  - "Write" benchmarks in many programming environments on different multicore architectures
  - Compare performance/watt and relative effort to serial C
- Conclusions
  - For homogeneous architectures C/C++ using threads or parallel arrays has highest productivity
  - For heterogeneous architectures C/C++ using hierarchical arrays has highest productivity